Racket Programming Assignment #4:

Recursive List Processing &

Higher Order Functions

Thomas Moskal

CSC 344 – Professor Graci

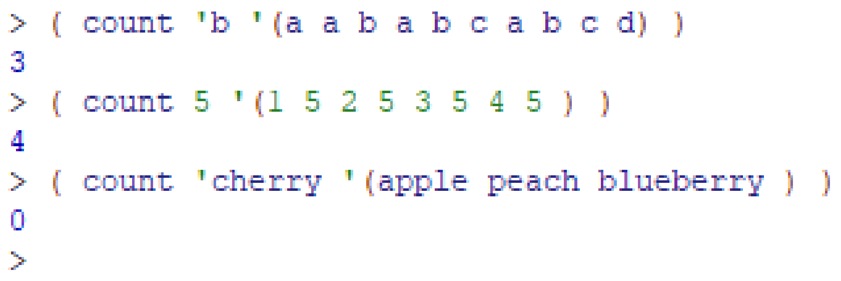
Due date: 4/1/22

Learning Abstract:

This assignment is an accumulation of our experiences in Racket Programming. In it we use skills we practiced in earlier assignments like recursion, lambda functions and working with sets. The assignment itself builds on itself and accumulates functions that are used several times in the later tasks. Higher Order functions are used throughout the assignment, and we get to see the utility of them firsthand. Higher order functions paired with recursive list processing leads to efficient and moddable code.

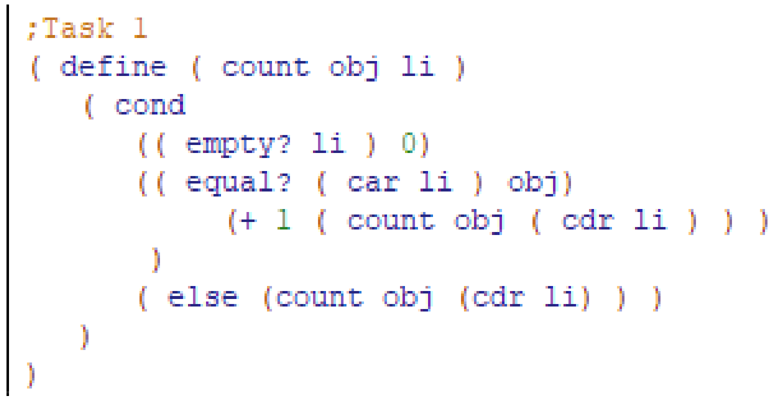
Task 1 – Count

Count Demo



```
> ( count 'b '(a a b a b c a b c d) )
3
> ( count 5 '(1 5 2 5 3 5 4 5 ) )
4
> ( count 'cherry '(apple peach blueberry ) )
0
>
```

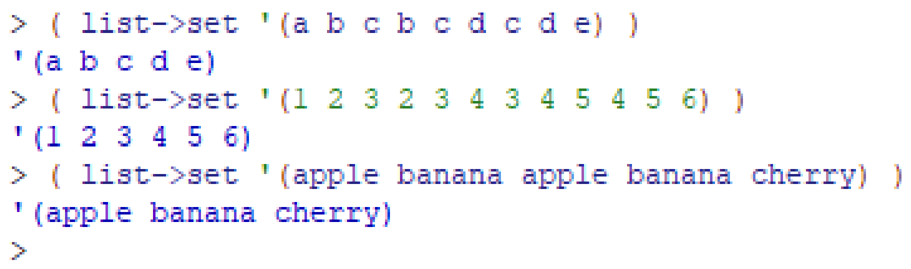
Count Code



```
;Task 1
( define ( count obj li )
   ( cond
      (( empty? li ) 0)
      (( equal? ( car li ) obj)
           (+ 1 ( count obj ( cdr li ) ) )
       )
      ( else (count obj (cdr li) ) )
   )
)
```

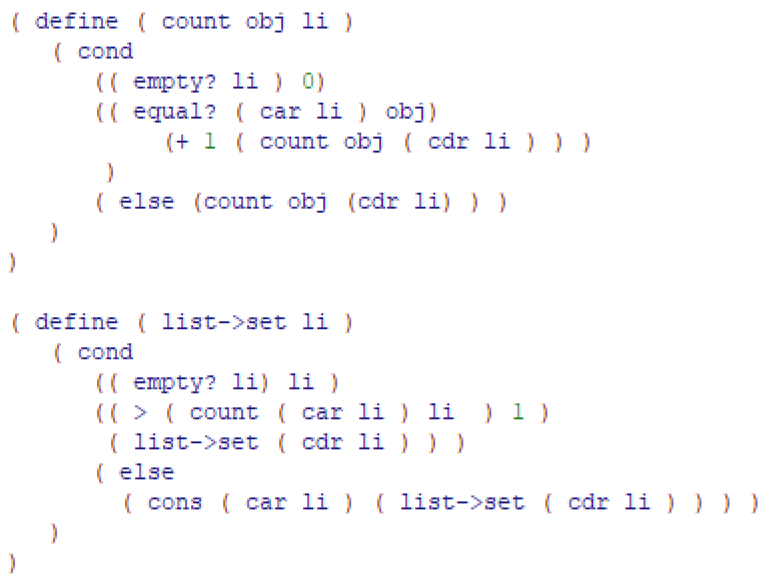
Task 2 – List-> set

List->set Demo



```
> ( list->set '(a b c b c d c d e) )
'(a b c d e)
> ( list->set '(1 2 3 2 3 4 3 4 5 4 5 6) )
'(1 2 3 4 5 6)
> ( list->set '(apple banana apple banana cherry) )
'(apple banana cherry)
>
```

List->set code



```
( define ( count obj li )
   ( cond
      (( empty? li ) 0)
      (( equal? ( car li ) obj)
          (+ 1 ( count obj ( cdr li ) ) )
       )
      ( else (count obj (cdr li) ) )
   )
)

( define ( list->set li )
   ( cond
      (( empty? li) li )
      (( > ( count ( car li ) li  ) 1 )
       ( list->set ( cdr li ) ) )
      ( else
        ( cons ( car li ) ( list->set ( cdr li ) ) ) )
   )
)
```

Task 3 – Association List Generator

Association List Generator Demo

```
>  ( a-list '(one two three four five) '(un deux trois quatre cinq) )
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> ( a-list '() '() )
'()
> ( a-list '( this ) '(that ) )
'((this . that))
> ( a-list '(one two three ) '( (1) (2 2) (3 3 3) ) )
'((one 1) (two 2 2) (three 3 3 3))
>
```

Association List Generator Code

```
( define ( a-list li1 li2 )
   ( cond
      (( empty? li1 ) cons li1 li2)
      (else
        ( cons (cons (car li1 ) ( car li2 ) ) (a-list ( cdr li1 ) ( cdr li2 ) ) ) )
    )
)
```

Task 4 – Assoc

Assoc Demo

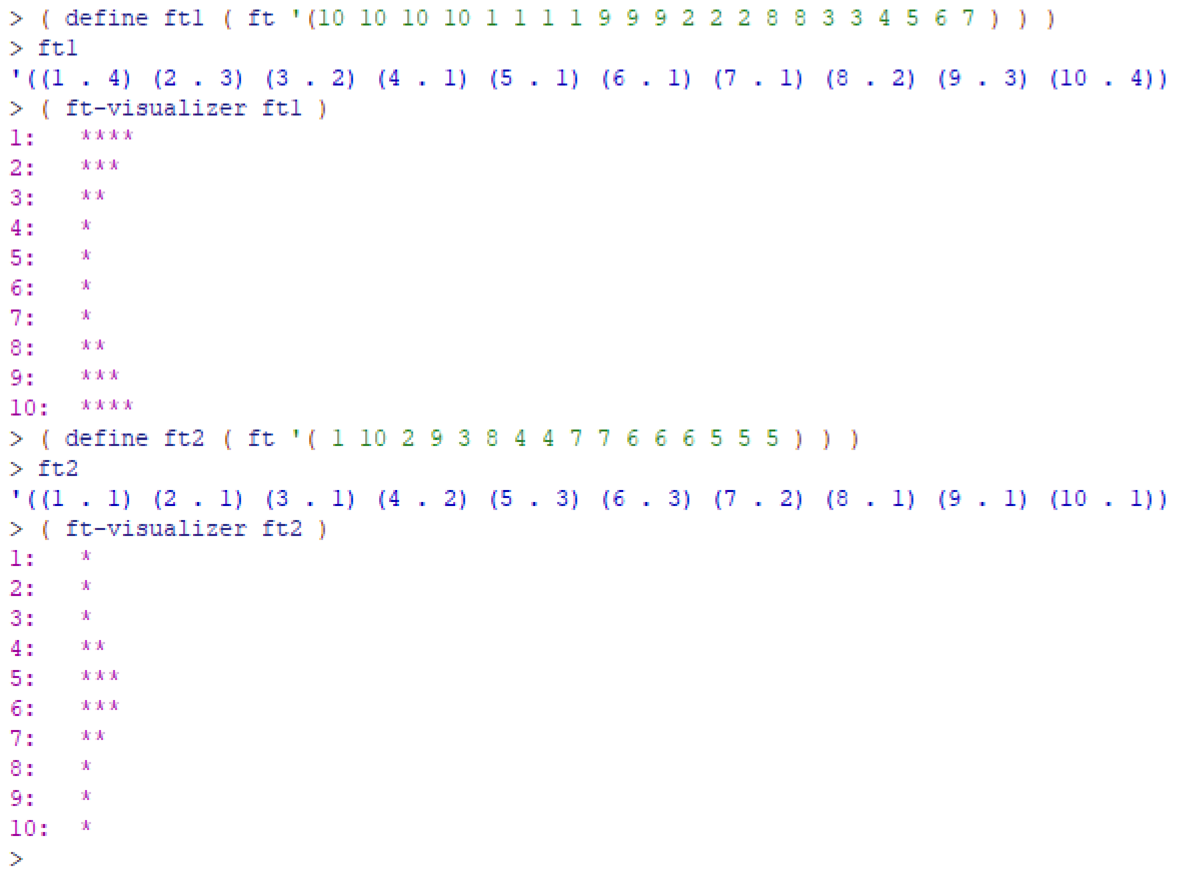
```
> ( define al1 ( a-list '(one two three four ) '(un deux trois quatre ) ) )
> ( define al2 ( a-list '(one two three) '( (1) (2 2) (3 3 3) ) ) )
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> ( assoc 'two al1 )
'(two . deux)
> ( assoc 'five al1 )
'()
> ( assoc 'three al2 )
'(three 3 3 3)
> ( assoc 'four al2 )
'()
>
```

Assoc Code

```
( define ( a-list li1 li2 )
   ( cond
      (( empty? li1 ) cons li1 li2)
      (else
        ( cons (cons (car li1 ) ( car li2 ) ) (a-list ( cdr li1 ) ( cdr li2 ) ) ) )
    )
)

( define ( assoc obj ali )
   (cond
     (( empty? ali ) '())
     (( equal? obj ( caar ali ) ) ( car ali ) )
     ( else
        ( assoc obj ( cdr ali ) ) )
   )
)
```

Task 5 – Frequency Table

Ft-Demo



```
> ( define ft1 ( ft '(10 10 10 10 1 1 1 1 9 9 9 2 2 2 8 8 3 3 4 5 6 7 ) ) )
> ft1
'((1 . 4) (2 . 3) (3 . 2) (4 . 1) (5 . 1) (6 . 1) (7 . 1) (8 . 2) (9 . 3) (10 . 4))
> ( ft-visualizer ft1 )
1:   ****
2:   ***
3:   **
4:   *
5:   *
6:   *
7:   *
8:   **
9:   ***
10:  ****
> ( define ft2 ( ft '( 1 10 2 9 3 8 4 4 7 7 6 6 6 5 5 5 ) ) )
> ft2
'((1 . 1) (2 . 1) (3 . 1) (4 . 2) (5 . 3) (6 . 3) (7 . 2) (8 . 1) (9 . 1) (10 . 1))
> ( ft-visualizer ft2 )
1:   *
2:   *
3:   *
4:   **
5:   ***
6:   ***
7:   **
8:   *
9:   *
10:  *
>
```

Ft-Code

```
( define ( list->set li )
   ( cond
      (( empty? li) li )
      (( > ( count ( car li ) li  ) 1 )
       ( list->set ( cdr li ) ) )
     ( else
        ( cons ( car li ) ( list->set ( cdr li ) ) ) )
   )
)

( define ( count obj li )
   ( cond
      (( empty? li ) 0)
      (( equal? ( car li ) obj)
           (+ 1 ( count obj ( cdr li ) ) )
       )
      ( else (count obj (cdr li) ) )
   )
)

( define ( a-list li1 li2 )
   ( cond
      (( empty? li1 ) cons li1 li2)
      (else
        ( cons (cons (car li1 ) ( car li2 ) ) (a-list ( cdr li1 ) ( cdr li2 ) ) ) )
    )
)

( define ( ft the-list )
   ( define the-set ( list->set the-list ) )
   ( define the-counts ( map ( lambda (x) ( count x the-list ) ) the-set ) )
   ( define association-list (a-list the-set the-counts ) )
   ( sort association-list < #:key car )
)

( define (ft-visualizer ft )
   ( map pair-visualizer ft )
   ( display "" )
)

( define ( pair-visualizer pair )
   ( define label
      ( string-append ( number->string ( car pair ) ) ":" ) )
   ( define fixed-size-label ( add-blanks label ( - 5 ( string-length label ) ) ) )
   ( display fixed-size-label )
   ( display
     (foldr string-append "" ( make-list ( cdr pair ) "*" ) ) )
   ( display "\n" )
)

( define ( add-blanks s n )
   ( cond
      ( ( = n 0 ) s )
      ( else ( add-blanks ( string-append s " " ) ( - n 1 ) ) )
   )
)
```

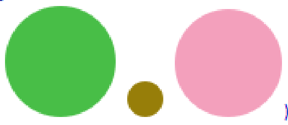
Frequency Table Q&A

1) List->set, count & a-list
2) (lambda (x) (count x the-list)) – counts the amount of the current object in the list.
3) The map application applies a procedure of n arguments to the elements of n lists. In the ft function map takes in two parameters, the lambda function (argument) and the-set (lists). The lambda argument itself only takes one parameter (and a set of instructions).
4) Getting the function to work with map.
5) Association-list
6) $:key <word> in this case, car.
7) Pair-visualizer
8) There would have been too many parameters for map & count to properly work together in the ft function.
9) String-append
10) No.
11) To avoid the #void
12) Stem and leaf plot is the data structure used. Visually, it is efficient in showing data, you can see statistics in it like a normal or bimodal distribution.
13) Yes.
14) Easily collecting and visualizing data by just using a few well thought out functions.
15) What would be the best way to use this code for statistical analysis?

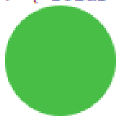
Task 6 – Generate List

Generate List Demo

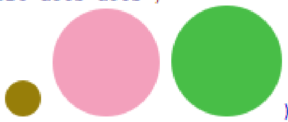
```
> ( generate-list 10 roll-die )
'(3 6 1 6 2 2 3 3 3 6)
> ( generate-list 20 roll-die )
'(3 3 6 4 6 6 4 1 3 1 4 3 1 5 3 1 6 6 2 3)
> ( generate-list 12
     ( lambda () ( list-ref '( red yellow blue ) ( random 3 ) ) )
)
'(red blue blue blue red red red red blue blue blue red)
> ( define dots ( generate-list 3 dot ) )
> dots
```



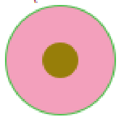
```
(list                      )
> ( foldr overlay empty-image dots )
```



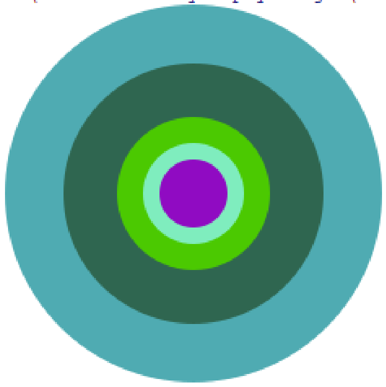
```
> ( sort-dots dots )
```



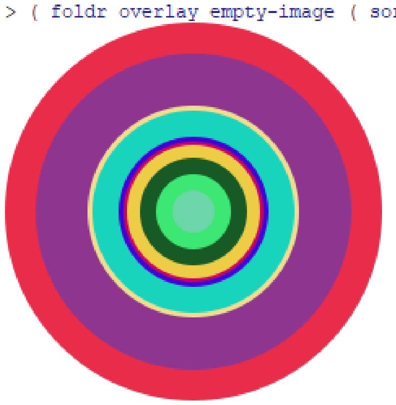
```
(list                      )
> ( foldr overlay empty-image ( sort-dots dots ) )
```



```
> ( define a ( generate-list 5 big-dot ) )
> ( foldr overlay empty-image ( sort-dots a ) )
```



```
> ( define b ( generate-list 10 big-dot ) )
> ( foldr overlay empty-image ( sort-dots b ) )
```



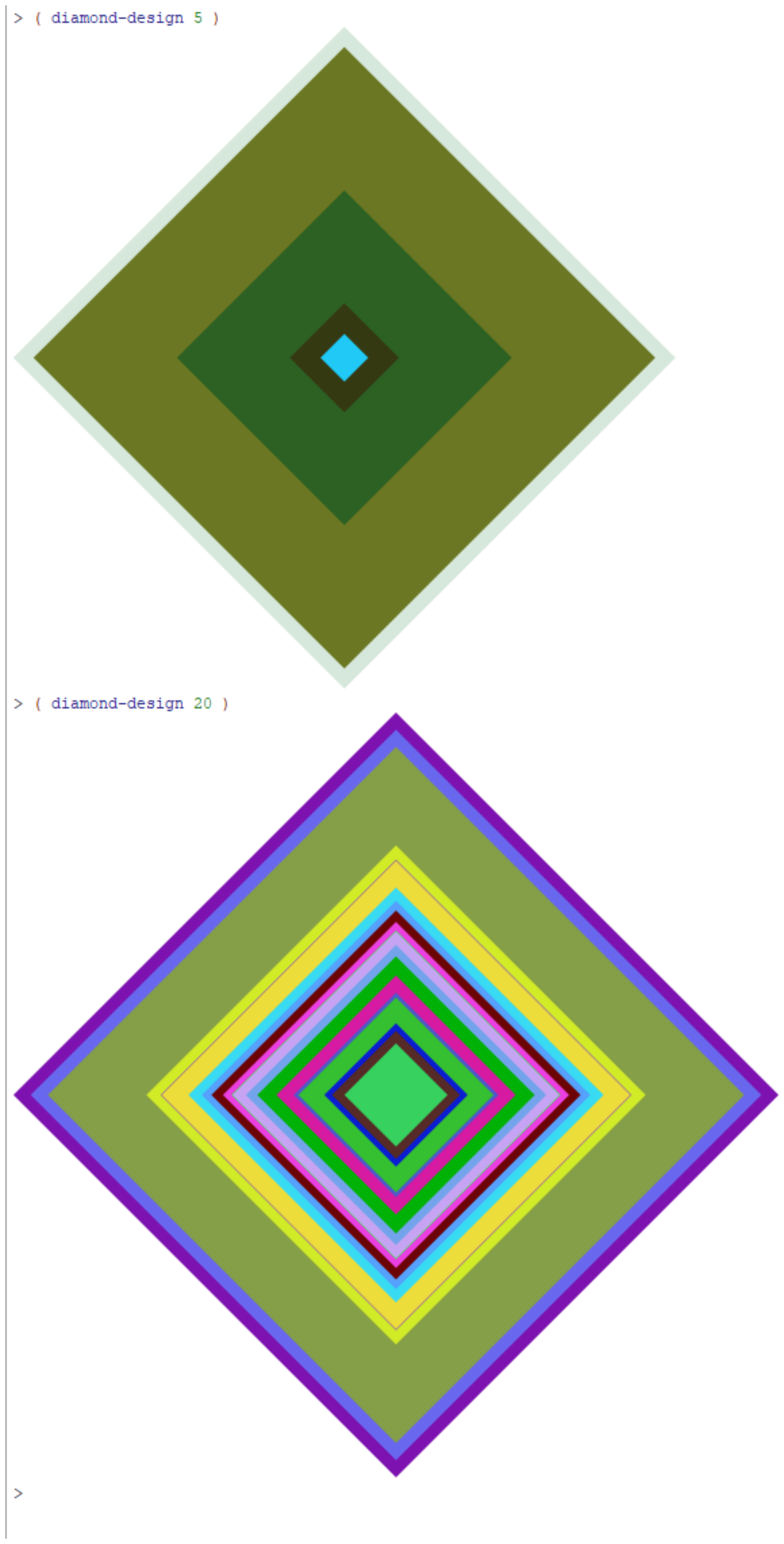
```
>
```

Generate List Code

```
( require 2htdp/image )

( define ( roll-die ) ( + ( random 6 ) 1 ) )

( define ( dot )
   ( circle ( + 10 ( random 41 ) ) "solid" ( random-color ) )
)

( define ( big-dot )
   ( circle ( + 10 ( random 120 ) ) "solid" ( random-color ) )
)

( define ( random-color )
   ( color ( random 256 ) ( random 256 ) ( random 256 ) )
)

( define ( sort-dots loc )
   ( sort loc #:key image-width < )
)

( define ( generate-list n funct )
   ( cond
      (( = n 0) '() )
      (( > n 0 )
         ( cons ( funct ) ( generate-list (- n 1 ) funct ) )
      )
   )
)
```

Task 7 – The Diamond

The Diamond Demo

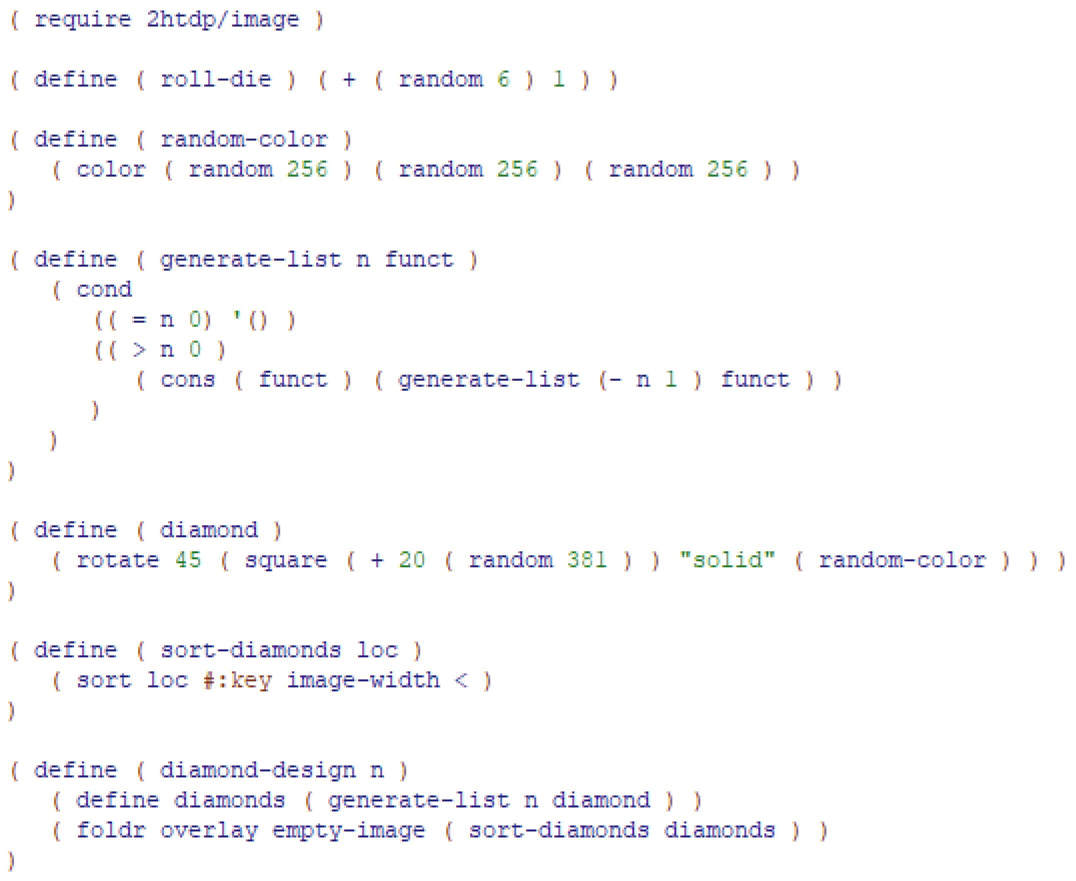


```
> ( diamond-design 5 )
```

```
> ( diamond-design 20 )
```

```
>
```

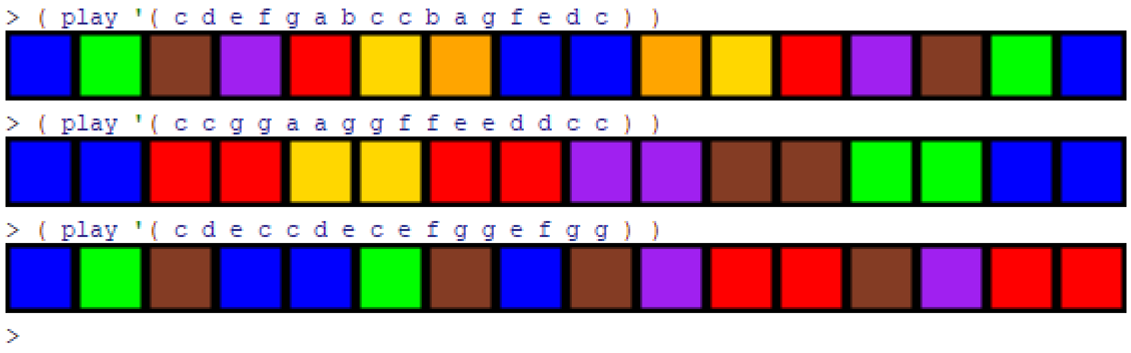
<u>The Diamond Code</u>



```
( require 2htdp/image )

( define ( roll-die ) ( + ( random 6 ) 1 ) )

( define ( random-color )
   ( color ( random 256 ) ( random 256 ) ( random 256 ) )
)

( define ( generate-list n funct )
   ( cond
      (( = n 0) '() )
      (( > n 0 )
         ( cons ( funct ) ( generate-list (- n 1 ) funct ) )
      )
   )
)

( define ( diamond )
   ( rotate 45 ( square ( + 20 ( random 381 ) ) "solid" ( random-color ) ) )
)

( define ( sort-diamonds loc )
   ( sort loc #:key image-width < )
)

( define ( diamond-design n )
   ( define diamonds ( generate-list n diamond ) )
   ( foldr overlay empty-image ( sort-diamonds diamonds ) )
)
```

Task 8 – Chromesthetic Renderings

Chromesthetic Rengerings Demo



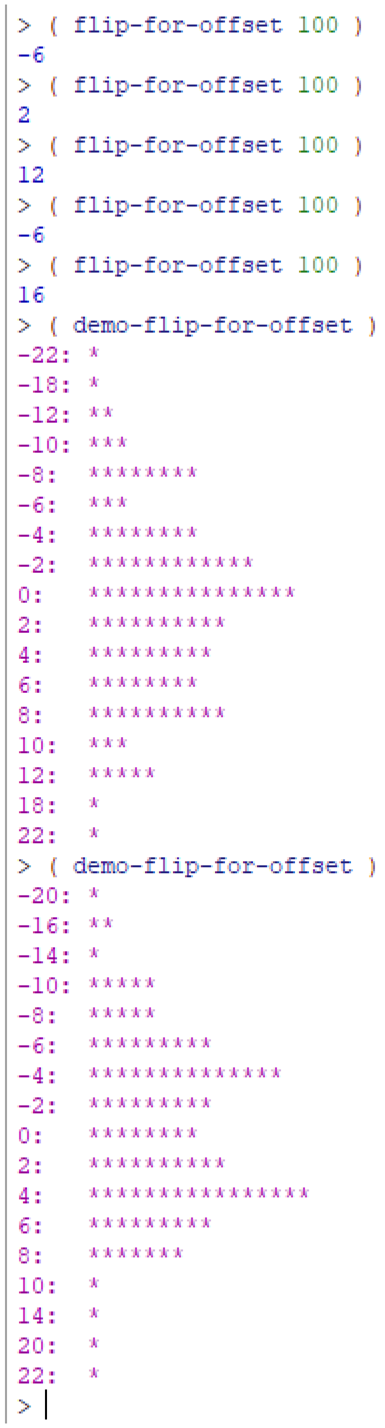
```
> ( play '( c d e f g a b c c b a g f e d c ) )
> ( play '( c c g g a a g g f f e e d d c c ) )
> ( play '( c d e c c d e c e f g g e f g g ) )
>
```

Chromesthetic Rengerings Code

```
(require 2htdp/image )

( define ( a-list lil li2 )
   ( cond
      (( empty? lil ) cons lil li2)
      (else
        ( cons (cons (car lil ) ( car li2 ) ) (a-list ( cdr lil ) ( cdr li2 ) ) ) )
     )
)

( define pitch-classes '( c d e f g a b ) )
( define color-names '( blue green brown purple red yellow orange ) )

( define ( box color )
   ( overlay
     (square 30 "solid" color )
     ( square 35 "solid" "black" )
    )
)

( define boxes (list
          ( box "blue" )
          ( box "green" )
          ( box "brown" )
          ( box "purple" )
          ( box "red" )
          ( box "gold" )
          ( box "orange" )
          )
)

( define pc-a-list ( a-list pitch-classes color-names ) )
( define cb-a-list ( a-list color-names boxes ) )

( define ( pc->color pc )
   ( cdr ( assoc pc pc-a-list ) )
)

( define ( color->box color )
   ( cdr ( assoc color cb-a-list ) )
)


( define ( play li )
   ( define colors ( map ( lambda ( x ) ( pc->color x ) ) li ) )
   ( define boxes ( map ( lambda ( x ) ( color->box x ) ) colors ) )
   ( foldr beside empty-image boxes )
)
```

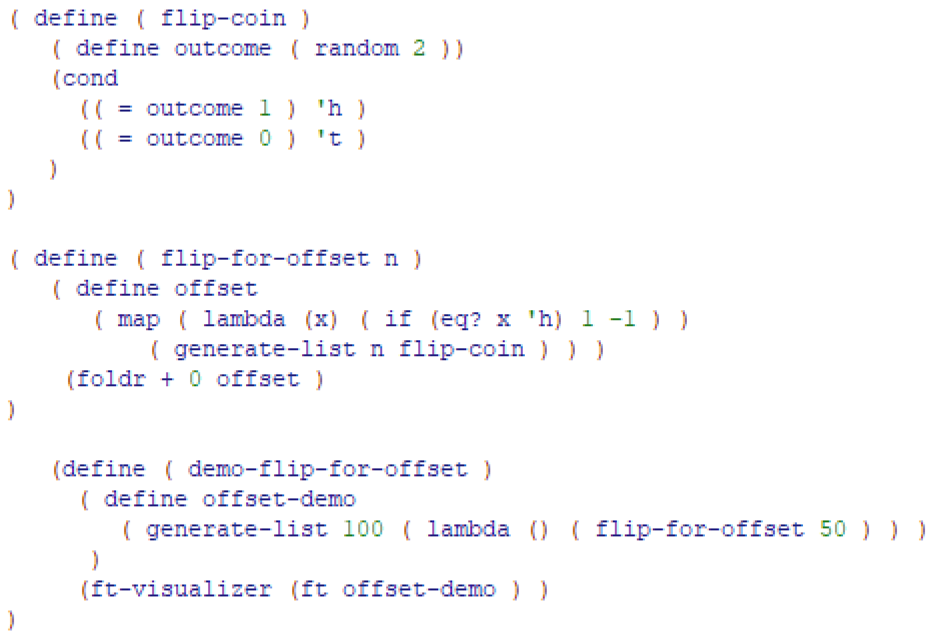
Task 9 – Transformation of a Recursive Sampler

Transformation of a Recursive Sampler Demo



```
> ( flip-for-offset 100 )
-6
> ( flip-for-offset 100 )
2
> ( flip-for-offset 100 )
12
> ( flip-for-offset 100 )
-6
> ( flip-for-offset 100 )
16
> ( demo-flip-for-offset )
-22: *
-18: *
-12: **
-10: ***
-8:  ********
-6:  ***
-4:  ********
-2:  ************
0:   ***************
2:   **********
4:   *********
6:   ********
8:   **********
10:  ***
12:  *****
18:  *
22:  *
> ( demo-flip-for-offset )
-20: *
-16: **
-14: *
-10: *****
-8:  *****
-6:  *********
-4:  **************
-2:  *********
0:   ********
2:   **********
4:   ****************
6:   *********
8:   *******
10:  *
14:  *
20:  *
22:  *
>
```

Transformation of a Recursive Sampler Code



```
( define ( flip-coin )
   ( define outcome ( random 2 ))
   (cond
     (( = outcome 1 ) 'h )
     (( = outcome 0 ) 't )
   )
)

( define ( flip-for-offset n )
   ( define offset
      ( map ( lambda (x) ( if (eq? x 'h) 1 -1 ) )
          ( generate-list n flip-coin ) ) )
    (foldr + 0 offset )
)

   (define ( demo-flip-for-offset )
     ( define offset-demo
        ( generate-list 100 ( lambda () ( flip-for-offset 50 ) ) )
      )
     (ft-visualizer (ft offset-demo ) )
)
```

Task 10 – Blood Pressure Trend Visualizer

Blood Pressure Trend Visualizer Demo

```
> ( sample 120 )
126
> ( sample 80 )
90
>  ( data-for-one-day 110 )
'(131 87)
>  ( data-for-one-day 110 )
'(123 89)
>  ( data-for-one-day 110 )
'(119 99)
>  ( data-for-one-day 90 )
'(105 63)
> ( data-for-one-day 90 )
'(101 71)
> ( data-for-one-day 90 )
'(113 79)
> ( data-for-one-week 110 )
'((127 85) (129 95) (135 97) (143 97) (125 83) (131 93) (133 97))
> ( data-for-one-week 100 )
'((118 82) (122 86) (116 80) (126 90) (122 80) (126 86) (120 78))
> ( data-for-one-week 90 )
'((115 65) (113 75) (117 71) (121 73) (105 67) (107 63) (107 77))
> ( define getting-worse '(95 98 100 102 105) )
> ( define getting-better '(105 102 100 98 95) )
> ( generate-data getting-worse )
'(((128 64) (122 66) (124 60) (124 64) (124 80) (110 68) (114 74))
  ((121 81) (123 73) (131 83) (131 79) (109 73) (113 99) (133 87))
  ((122 82) (124 64) (110 80) (118 80) (118 76) (102 72) (128 72))
  ((127 89) (129 85) (121 81) (127 91) (117 81) (127 79) (121 87))
  ((123 73) (115 81) (103 83) (125 73) (135 85) (109 89) (109 83)))
> ( generate-data getting-better )
'(((133 93) (123 77) (131 73) (117 83) (133 71) (121 79) (129 97))
  ((127 81) (125 71) (115 83) (133 73) (123 85) (129 79) (111 79))
  ((114 72) (88 80) (102 76) (110 92) (130 70) (122 76) (128 76))
  ((117 85) (105 77) (113 79) (121 81) (97 75) (111 85) (127 75))
  ((110 86) (102 74) (118 76) (128 70) (114 72) (114 70) (108 74)))
> ( one-day-visualization '(125 83) )
🔴

> ( one-day-visualization '(125 78) )
🟡

> ( one-day-visualization '(116 87) )
🟠

> ( one-day-visualization '(114 75) )
🔵

> ( define bad-week ( data-for-one-week 110 ) )
> ( define good-week ( data-for-one-week 90 ) )
> bad-week
'((127 89) (129 109) (147 85) (143 85) (123 75) (123 87) (131 83))
> ( one-week-visualization bad-week )
(🔴 🔴 🔴 🔴 🟡 🔴 🔴)
> good-week
'((105 63) (105 55) (119 77) (123 67) (109 71) (119 77) (107 71))
> ( one-week-visualization good-week )
(🔵 🔵 🔵 🟡 🔵 🔵 🔵)
> ( define bp-data ( generate-data '(110 105 102 100 98 95 90 ) ) )
> ( bp-visualization bp-data )
(🔴 🔴 🟡 🔴 🔴 🔴 🔴)
(🟠 🔴 🔴 🟡 🔴 🔴 🟡)
(🟡 🟠 🔵 🔴 🔴 🔴 🔴)
(🟡 🔴 🔴 🟠 🔵 🟡 🟠)
(🟠 🟠 🟡 🟠 🔴 🟠 🟡)
(🔵 🟡 🔵 🔴 🔵 🔵 🔵)
(🔵 🔵 🔵 🔵 🔵 🔵 🔵)
>
```

Blood Pressure Trend Visualizer Code

```
( define ( sample cardio-index )
   ( + cardio-index ( flip-for-offset ( quotient cardio-index 2 ) ) )
)

( define ( data-for-one-day middle-base )
   ( list
     ( sample ( + middle-base 20 ) )
     ( sample ( - middle-base 20 ) )
   )
)

( define ( data-for-one-week middle-base )
   ( generate-list 7
                   (lambda () ( data-for-one-day middle-base ) )
    )
)

( define ( generate-data base-sequence )
   ( map data-for-one-week base-sequence )
)

( define ( color-dot c )
   ( circle 10 "solid" c )
)

( define ( one-day-visualization bp )
   ( define h ( car bp ) )
   ( define l ( cadr bp ) )
   ( cond
      ( ( and ( >= h 120 ) ( >= l 80 ) ) ( color-dot "red" ) )
      ( ( and ( >= h 120 ) ( < l 80 ) ) ( color-dot "gold" ) )
      ( ( and ( < h 120 ) ( >= l 80 ) ) ( color-dot "orange" ) )
      ( ( and ( < h 120 ) ( < l 80 ) ) ( color-dot "blue" ) )
    )
)

( define ( one-week-visualization bp )
   ( display ( map one-day-visualization bp ) )
)

( define ( bp-visualization bp )
   ( cond
      ( ( empty? bp ) (display "" ) )
      (else
        ( one-week-visualization ( car bp ) )
        ( display "\n" )
        ( bp-visualization ( cdr bp ) )
      )
   )
)
```